Skyblocking for Entity Resolution

Jingyu Shao, Qing Wang, and Yu Lin

Research School of Computer Science
College of Engineering and Computer Science
The Australian National University
Canberra, ACT, 2601, Australia.

Abstract

In this paper, we introduce a novel framework for entity resolution blocking, called skyblocking, which aims to learn scheme skylines. In this skyblocking framework, each blocking scheme is mapped as a point to a multi-dimensional scheme space where each blocking measure represents one dimension. A scheme skyline contains blocking schemes that are not dominated by any other blocking schemes in the scheme space. To efficiently learn scheme skylines, two challenges exist: one is the class imbalance problem and the other is the search space problem. We tackle these two challenges by developing an active sampling strategy and a scheme extension strategy. Based on these two strategies, we develop three scheme skyline learning algorithms for efficiently learning scheme skylines under a given number of blocking measures and within a label budget limit. We experimentally verify that our algorithms outperform the baseline approaches in all of the following aspects: label efficiency, blocking quality and learning efficiency, over five real-world datasets.

Keywords: Entity resolution, blocking scheme, active learning, skyline.

1. Introduction

Entity Resolution (ER) is of great importance in many applications, such as matching product records from different online stores for customers, detecting people's financial status for national security, or analyzing health conditions from different medical organizations [13, 21]. Generally, entity resolution refers to the process of identifying records which represent the same real-world entity from one or more datasets [48]. Common ER approaches require the calculation of similarity between records in order to classify record pairs as matches or non-matches. However, with the size of a dataset growing, the similarity computation time for records increases quadratically. For example, given a dataset D, the total number of record pairs to be compared is $\frac{|D|*(|D|-1)}{2}$ (i.e., each record is paired with all other records in D). To eliminate the unnecessary computation, blocking techniques are widely applied, which can group potentially matched records into the same block [47]. and restrict the comparison to only occurring between records in the same block. In doing so, blocking can reduce the number of compared record pairs to no more than $\frac{m*(m-1)}{2} * n$, where m is the number of records in the largest block and n is the number of blocks.

In past years, a good number of techniques have been proposed for blocking [17, 47, 21, 38, 39, 44]. Among these techniques, using blocking schemes is an efficient and declarative way to generate blocks. Intuitively, a blocking scheme takes records from a dataset as input, and groups the records using a logical combination of blocking predicates, where each blocking predicate specifies an attribute and its corresponding function. Learning a blocking scheme is thus the process of deciding which attributes are chosen for blocking, what the corresponding methods are used to compare values in attributes, and how different attributes and methods are logically combined so that desired blocks can be generated to satisfy the needs of an application. Compared with blocking techniques that consider data at the instance level [17, 21], blocking schemes have several advantages: (1) They only require to decide what metadata, such as attributes and the corresponding methods, is needed, rather than what data from individual records are selected; (2) They provide a more human readable description for how attributes and methods are involved in blocking; and (3) They enable more natural and effective interaction for blocking across heterogeneous datasets.

However, ER applications often involve multi-criteria analysis varying preferences of blocking schemes. More specifically, given a scheme space that contains a large number of possible blocking schemes and a collection of criteria for choosing blocking schemes such as: pair completeness (PC), pair quality (PQ) and reduction ratio (RR) [14], how can we identify the *most preferred* blocking scheme? Ideally, a good blocking scheme should yield blocks that minimize the number of record pairs to compare, while still preserving true matches as many as possible, i.e. optimizing all criteria simultaneously. Unfortunately, the criteria for selecting blocking schemes are often competing with each other. For example, PC and PQ are negatively correlated in many applications, as well as RR and PQ [14]. In Fig. 1, we can see that, a blocking scheme with an improved PC leads to a decrease in PQ, and conversely, a blocking scheme with an increase in PQ may have a decease in PC. Depending on specific application requirements, differ-

Blocking scheme	PC	PQ
s_1	0.13	0.76
s_2	0.31	0.99
s_3	0.58	0.76
s_4	0.84	0.40
s_5	0.86	0.50
...	...	...

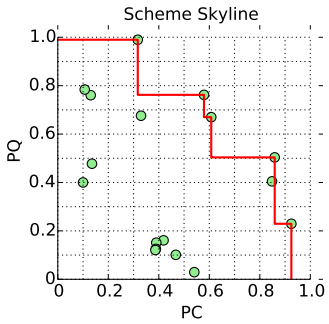

Figure 1: An example for a scheme skyline (schemes in red line) from Cora Dataset, where blocking schemes (green points) are presented in a 2-dimensional space and their corresponding PC and PQ values are shown in the table.

ent blocking schemes can be considered as being "preferred" in different applications.

Thus, to effectively learn a blocking scheme that is optimal under one or more criteria, previous works specified various constraints in the learning process [26]. For example, some approaches [7, 32] aimed to learn a blocking scheme that can maximize both reduction ratio of record pairs regardless of whether blocking is used, and coverage of matched pairs. Some others [5, 24] targeted to find a blocking scheme that can generate blocks containing almost all matched record pairs with a minimal number of non-matched record pairs. Shao and Wang [44] considered to select a blocking scheme with a minimal number of non-matched pairs constrained under a given coverage rate for matched pairs. However, setting such constraints perfectly is a challenging task because constraints may vary in different applications and it is often *unknown* which constraint is appropriate for a specific application. If a constraint is set strictly, no blocking scheme can be learned; on the other hand, if a constraint is set loosely, a learned blocking scheme is not necessarily the most preferred one. Thus, the question that naturally arises is: can we efficiently identify all possible most preferred blocking schemes in terms of a given set of selection criteria for blocking?

To answer this question, we propose a scheme skyline learning framework for blocking, called *skyblocking*, which incorporates skyline techniques into the active learning process of blocking schemes on the skyline. The target of skyblocking is to find a range of blocking schemes (as shown in Figure 1) under different user preferences and within a limited label budget, and each of such blocking schemes is optimal with respect to one or more selection criteria.

Applications Entity resolution is widely applied in many applications [13]. Users may have different preference on blocking schemes to deal with different applications, and thus to achieve ER results from various perspectives. For example,

- *Crime investigation:* In crime investigation, when individuals are investigated for a crime, it is necessary to identify as many candidates as possible so that the criminal will not be missed out. In this case, blocking schemes with high PC values would be preferred.
- *Medical study:* When studying the medical conditions of patients, we would need to identify patients that exactly correspond to certain medical conditions. In this case, blocking schemes with relatively high PQ values would be desired, because they can help match the patients and the medical conditions so as to diagnose patients that are necessarily included under study.

Challenges To the best of our knowledge, no scheme skyline learning approach has been previously proposed for entity resolution. Traditionally, skyline techniques have been widely used for database queries, called skyline queries [23, 10, 42]. The result of a skyline query consists of multi-dimensional points for which there is no other point having better values in all the dimensions [9]. However, applying skyline queries technique to learning scheme skylines is a non-trivial task. Different from skyline queries in the database context [10], in which the dimensions of a skyline correspond to attributes and points correspond to records, scheme skylines in our work have the dimensions corresponding to selection criteria for blocking, and points corresponding to blocking schemes in this multi-dimensional space, called *scheme space*. Learning scheme skylines becomes challenging, due to the following unique characteristics of blocking schemes.

Firstly, it is hard to obtain labels in real-world ER applications. Particularly, it is well-known that match and non-match labels for entity resolution are often highly imbalanced, called *the class imbalance problem* [20, 16, 49]. For example, given a dataset with two tables, each table containing 1,000 non-duplicate records, the total number of record pairs will be 1,000,000, but the number of true matches is no more than 1,000. The class imbalance ratio of this dataset is thus at most 1:1,000. This indicates that the probability of randomly selecting a matched pair from this dataset is 0.001. However, to find scheme skylines effectively, we need a training set that contains a sufficient number of matches. Hence, the first challenge we must address is how to select more matched samples within a limited label budget for blocking scheme learning in a scheme space.

Secondly, a scheme space can be very large, making an exhaustive search for blocking schemes on a skyline computationally expensive. As will be discussed in Section 4.5, if a blocking scheme is composed of at most n different blocking predicates, the number of all possible blocking schemes can be $O(2^{\binom{n}{\lfloor n/2 \rfloor}})$ asymptotically. Nevertheless, only a small portion of these blocking schemes are on a skyline. Thus, the second challenge we need to act on is how to design efficient skyline algorithms to explore a large scheme space efficiently for finding blocking schemes on a skyline. Enumerating through the entire scheme space by checking a blocking scheme each time is obviously not efficient in practice, and even infeasible for large datasets with hundreds or thousands of attributes.

Contributions To overcome the above challenges, in this work, we hierarchically learn a scheme skyline based on blocking predicates to avoid enumerating all possible blocking schemes. We also develop a balanced active sampling strategy, which aims to select informative samples within a limited budget of

labels. In summary, the contributions of this paper are as follows.

- We formulate a novel scheme skyline learning problem for entity resolution. Solving this problem would lead to generating a range of optimal blocking schemes with respect to different blocking criteria and thus enable users to choose their preferred blocking schemes.
- We propose three novel scheme skyline learning algorithms to efficiently learn scheme skylines within a limited label budget. In these algorithms, we tackle the class imbalance problem by first converting this problem into the balanced sampling problem and then developing an active sampling strategy to actively select informative samples. We also develop a scheme extension strategy for efficiently identifying schemes that are possibly on a skyline in order to reduce the search space and label cost used in the learning process.
- We have evaluated the efficiency and effectiveness of our scheme skyline learning algorithms over five real-world datasets. The experimental results show that our algorithms outperform the baseline approaches in all of the following aspects: label efficiency, blocking quality and learning efficiency.

Outline The rest of the paper is structured as follows. Section 2 introduces the notations used in the paper and the problem definition. Section 3 discusses our active sampling strategy to the class imbalance problem. Three skyline algorithms for learning scheme skylines are presented in Section 4. Section 5 discusses our experimental results, and Section 6 introduces the related work. We conclude the paper in Section 7.

2. Problem Definition

Let D be a dataset consisting of records. Each record $r \in D$ is associated with a set of attributes A, and each attribute $a \in A$ has a domain $Dom(a)$. We use $r.a$ to refer to the value of attribute a in a record r. A *blocking function* $h : Dom(a) \times Dom(a) \rightarrow \{0, 1\}$ takes a pair of attribute values from $Dom(a)$ as input and returns a value in $\{0, 1\}$ as output. A *blocking predicate* $\langle a, h \rangle$ is a pair of a blocking attribute a and a blocking function h. Given a pair of records $\langle r_i, r_j \rangle$, a blocking predicate $\langle a, h \rangle$ returns *true* if $h(r_i.a, r_j.a) = 1$ and returns *false* if $h(r_i.a, r_j.a) = 0$.

Example 1. *Consider the dataset example in Table 1 and suppose that we have a blocking predicate* ⟨*Authors, Same-soundex*⟩.

In Table 1, the record r_1 *has "Gale" in the attribute Authors, while the records* r_2 *and* r_3 *have "Gaile" in the attribute Authors. The soundex value of both "Gale" and "Gaile" is G4. The other records* r_4 *and* r_5 *have "Johnson" in the attribute Authors and the soundex value of "Johnson" is 75. Hence, if we consider the pair of records* $\langle r_1, r_2 \rangle$*, this blocking predicate returns* true *because Same-soundex*$(Gale, Gaile) = 1$. *However, for the pair of records* $\langle r_1, r_4 \rangle$*, the blocking predicate returns* false *because Same-soundex*$(Gale, Johnson) = 0$.

Table 1: A bibliographic dataset example including three attributes: Title, Authors and PublicationYear.

ID	Title	Authors	PublicationYear
r_1	An active learning ...	Gale	2003
r_2	An active learning ...	Gaile	2003
r_3	Entity resolution for ...	Gaile	2006
r_4	An active learning ...	Johnson	2003
r_5	Active learning blocking ...	Johnson	2003

Given a set of blocking predicates P, the *feature vector* of a record pair $\langle r_i, r_j \rangle$ is a tuple $x = \langle v_1, v_2, ...v_{|P|} \rangle$, where each v_k $(k = 1, \ldots, |P|)$ is either 1 or 0, describing whether the corresponding blocking predicate in P returns true or false, respectively. For each feature vector x, a *human oracle* ζ is used to provide a label $y \in \{M, N\}$. If $y = M$, it indicates that $\langle r_i, r_j \rangle$ refers to the same entity (i.e. *a match*), and analogously, $y = N$ indicates that $\langle r_i, r_j \rangle$ refers to two different entities (i.e. *a non-match*). The human oracle ζ is associated with a budget limit $budget(\zeta) \geq 0$, which indicates the total number of labels ζ can provide. A *training set* $T = (X, Y)$ consists of a set of feature vectors $X = \{x_1, x_2, ..., x_{|T|}\}$ and their labels $Y = \{y_1, y_2, ..., y_{|T|}\}$.

A *(blocking) scheme* s is a disjunction of conjunctions of blocking predicates (i.e. in the disjunctive normal form). A blocking scheme is called a *n-ary blocking scheme* if it contains n distinct blocking predicates. Given a blocking scheme $s = s_1 \vee s_2 ... \vee s_n$, where each s_i $(i = 1, \ldots, n)$ is a conjunction of blocking predicates, we can generate a set of pairwise disjoint blocks $B_s = \{b_1, \ldots, b_{|B_s|}\}$, where $b_k \subseteq D$ $(k = 1, \ldots, |B_s|)$, $\bigcup_{1 \leq k \leq |B_s|} b_k = D$ and $\bigwedge_{1 \leq i \neq j \leq |B_s|} b_i \cap b_j = \emptyset$. Two records r_i and r_j are placed into the same block if there exists a conjunction of block predicates s_i in the given block scheme s such that the feature vector x of $\langle r_i, r_j \rangle$ contains 1 for each blocking predicate in s_i. We say $s(x) = true$ in this case; otherwise, $s(x) = false$.

Example 2. *Given* $s_1 =$ ⟨*Authors, Same-soundex*⟩∧⟨*Title, Same-value*⟩ *and* $s_2 =$ ⟨*Authors, Same-soundex*⟩ ∧ ⟨*PublicationYear, Same-value*⟩*,* $s = s_1 \vee s_2$ *is a 3-ary blocking scheme. This is because* s *contains three distinct blocking predicates, i.e.* ⟨*Authors, Same-soundex*⟩*,* ⟨*PublicationYear, Same-value*⟩*, and* ⟨*Title, Same-value*⟩*.*

By applying the blocking scheme s *on the records in Table 1, a set of blocks* $B_s = \{\{r_1, r_2\}, \{r_3\}, \{r_4, r_5\}\}$ *would be generated since* s *returns true for* $\langle r_1, r_2 \rangle$ *as well as* $\langle r_4, r_5 \rangle$*.*

Given a dataset D and a set of blocking schemes S over D, a *scheme space* is a d-dimensional space consisting of points in $[0, 1]^d$. Each point, denoted as $p(s)$, is associated with a blocking scheme $s \in S$ such that $p(s) = (p_1.s, p \ldots, p_d.s)$, where each $p_i.s$ indicates the value of s in the i-th dimension of this scheme space and $i \in [1, d]$. For example, in Fig. 1, each green point is associated with a blocking scheme in a 2-dimensional space.

Definition 1. *(*Dominance*) Given two blocking schemes* s_1 *and* s_2*, we say* s_1 dominates s_2*, denoted as* $s_1 \succ s_2$*, iff* $\forall i \in d, p_i.s_1 \geq p_i.s_2$ *and* $\exists j \in d, p_j.s_1 > p_j.s_2$*.*

Here, $p_i.s_1 \geq p_i.s_2$ indicates that the value of s_1 is larger than that of s_2 in the i-th dimension. Based on the notion of dominance between two blocking schemes, we define the notion of scheme skyline.

Definition 2. *(*Scheme skyline*) Given a dataset D and a set of blocking schemes S, a* scheme skyline *S^* over S is a subset of S, where each scheme $s \in S^*$ is not dominated by any other blocking scheme in S, i.e. $S^* = \{s \in S : \nexists s' \in (S - S^*), s' \succ s\}$.*

Without loss of generality, we will discuss a scheme space with $d = 2$ in the rest of this paper: one dimension indicates the PC values of blocking schemes and the other dimension indicates the PQ values of blocking schemes. Note that, it is possible to extend the dimensions of a scheme space by taking into account other measures for blocking schemes, such as reduction ratio (RR) and run time (RT) [28]. We will further discuss the properties of such measures later in Section 4.

Both PC and PQ are commonly used measures for blocking [14]. For a pair of records that are placed into the same block, we call it a *true positive* if it refers to a match; otherwise, it is a *false positive*. Similarly, a pair of records is called a *false negative* if it refers to a match but the records are placed into two different blocks. For convenience, we use $tp(s)$, $fp(s)$ and $fn(s)$ to denote the numbers of true positives, false positives and false negatives in blocks w.r.t. a blocking scheme s, respectively. The *pair completeness* (PC) of a blocking scheme s is the number of true positives $tp(s)$ divided by the total number of true matches, i.e. $tp(s) + fn(s)$, which measures the rate of true matches remained in blocks:

$$PC(s) = \frac{tp(s)}{tp(s) + fn(s)} \tag{1}$$

The *pair quality* (PQ) of a blocking scheme s is the number of true positives $tp(s)$ divided by the total number of record pairs that are placed into the same blocks, i.e. $tp(s) + fp(s)$, which measures the rate of true positives in blocks:

$$PQ(s) = \frac{tp(s)}{tp(s) + fp(s)} \tag{2}$$

We now define the problem of scheme skyline learning, which allows us to select desired blocking schemes from a scheme space depending upon which blocking criteria we prefer.

Definition 3. *(*Scheme skyline learning problem*) Let ζ be a human oracle, D be a dataset and S be a set of blocking schemes over D. Then the* scheme skyline learning problem *is to learn a scheme skyline S^* over S through actively selecting a training set T from D, such that $|T| \leq budget(\zeta)$ holds.*

3. Class Imbalance Learning

Previously, it has been reported that training data with imbalanced classes has impeded the performance of learning approaches in entity resolution [5, 7, 24]. Although the ratio of non-matches and matches may vary in different datasets, using random sampling can almost always generate much more non-matches than matches in training data, as illustrated in Fig. 2. To effectively learn scheme skylines, it is thus important to have a training set where matches and non-matches are balanced.

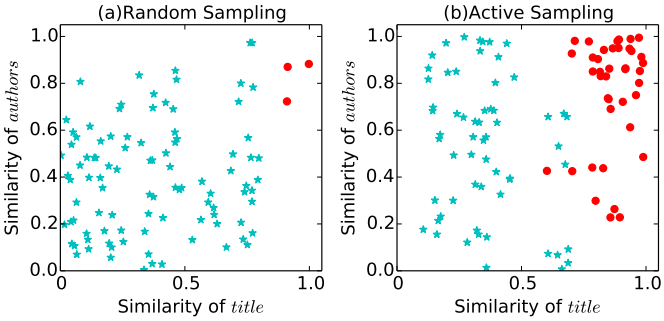

Figure 2: A comparison on the sample distribution of 100 samples from Cora dataset: (a) random sampling, and (b) active sampling (based on similarity of two attributes *title* and *authors*, where a red circle indicates a match sample and a blue triangle indicates a non-match sample.

In the following, we propose an active sampling strategy, aiming to generate a balanced training set by using only a small amount of labels. We observe that the correlation between the similarity of features (i.e., how similar the features of two records are) and the labels (i.e., $\{M, N\}$) can be leveraged to reduce the imbalance of matches and non-matches in sampling. More specifically, this observation is based on two empirical facts: (1) the more similar two records are, the higher probability they can be a match, as reported in previous works [1, 3]; (2) the similarity of some features may correlate more closely with matches than the similarity of other features. For example, in a bibliographic dataset, if we have 10 record pairs whose title are the same and another 10 record pairs whose publication year are the same, then it is likely that the former pairs refer to more matches in comparison with the latter pairs. Hence, by virtue of the correlation between the similarity of features and the match and non-match labels, a balanced set of similar and dissimilar records likely represents a training set with balanced matches and non-matches. To formulate this observation, we define the notion of balance rate in the following.

Definition 4. *(*Balance rate*) Let s be a blocking scheme and X a non-empty set of feature vectors, the* balance rate *of X in terms of s, denoted as $\gamma(s, X)$, is defined as:*

$$\frac{|\{x \in X | s(x) = true\}| - |\{x \in X | s(x) = false\}|}{|X|} \tag{3}$$

Conceptually, the balance rate describes how balanced X is by comparing the number of similar samples (i.e., $s(x) = true$) to the number of dissimilar samples (i.e., $s(x) = false$) in terms of s. The range of a balance rate is $[-1, 1]$. If $\gamma(s, X) = 1$, it means that all the samples in X are similar record pairs with regard to s, and conversely $\gamma(s, X) = -1$ means that all the samples in X are dissimilar record pairs. In these two cases, X is highly imbalanced. If $\gamma(s, X) = 0$, there is an equal number of similar and dissimilar record pairs, indicating that X is balanced.

Thus, based on the notion of balance rate, we convert the class imbalance problem into the balanced sampling problem as defined below.

Definition 5. *(*Balanced sampling problem*) Given a set of blocking schemes S and a label budget budget(ζ), the* balanced sampling problem *is to select a training set $T = (X, Y)$, where $|X| = |Y| = budget(\zeta)$, in order to:*

$$\textbf{\textit{minimize}} \sum_{s_i \in S} \gamma(s_i, X)^2 \tag{4}$$

For two different blocking schemes s_1 and s_2, they may have different balance rates over the same set of feature vectors X, i.e. $\gamma(s_1, X) \neq \gamma(s_2, X)$ is possible. The goal here is to find a training set that minimizes the balance rates in terms of a given set of blocking schemes. This process is called *Active Sampling*. The optimal case is $\gamma(s_i, X) = 0, \forall s_i \in S$ according to the objective function above, but sometimes it is impossible to achieve this in real world applications.

Example 3. *Consider the dataset example in Table 1 again, and let X refer to the set of all possible pairs of records from this table. Then, for a blocking scheme $s_1 = \langle$Title, Same-value$\rangle$, there are three similar samples out of all possible samples: $\langle r_1, r_2\rangle$, $\langle r_1, r_4\rangle$ and $\langle r_2, r_4\rangle$. That is, $\gamma(s_1, X) = 0.4$. For another blocking scheme $s_2 = \langle$Authors, Same-soundex$\rangle$ described in Example 1, we have four similar samples: $\langle r_1, r_2\rangle$, $\langle r_1, r_3\rangle$, $\langle r_2, r_3\rangle$ and $\langle r_4, r_5\rangle$, i.e., $\gamma(s_2, X) = 0.2$.*

Fig. 2 illustrates a comparison of sampling results by using random sampling and by using our active sampling strategy. We can see that, due to the existence of the class imbalance problem, the samples from random sampling are almost all non-matches, whereas the samples from our active sampling contain much more matches than the samples from random sampling.

4. Scheme Skyline Learning Approaches

Based on the active sampling strategy discussed in the previous section, we now propose three different algorithms to solve the scheme skyline learning problem. Since one of the key challenges faced by scheme skyline learning is to efficiently search for blocking schemes in a skyline from a large scheme space, we will first discuss an efficient scheme extension strategy. Then, we will introduce three skyline learning algorithms, and discuss their advantages and disadvantages. Finally, we will formally analyze the search and time complexity of our skyline learning algorithms.

4.1. Scheme Extension Strategy

Given a blocking scheme s, there are two kinds of possible extensions: conjunction and disjunction. Although iterating all the possible extensions is theoretically feasible, it is not efficient. Hence, one would expect that, if a conjunction or disjunction can be decided before extension, the searching space would be at least reduced by half. In line with this, we explore the monotonic property of measures for blocking. Let s_i and s_j be two blocking schemes. We have the following lemma.

Lemma 1.

$$PC(s_i) \leq PC(s_i \vee s_j) \tag{5}$$

$$PC(s_i) \geq PC(s_i \wedge s_j) \tag{6}$$

Proof 1. *Suppose that x is a true positive in B_{s_i}, then we would have $x \in B_{s_i} \cup B_{s_j} = B_{(s_i \vee s_j)}$. This is to say, true positives generated by either s_i or s_j must also be generated by $s_i \vee s_j$, and must contain all true positives generated by $s_i \wedge s_j$. Since we know the sum of true positives and false negatives is constant, which refers to the total number of matched pairs in a dataset, by the definition of PC, the lemma is proven.*

This lemma implies that, if a scheme generates blocks with a low PC value, we can extend it with disjunction in order to increase the PC value. On the contrary, if extending this scheme by conjunction, the PC value will be decreased.

Generally, if a measure of blocking [35] has a monotonic property in terms of the conjunction and disjunction of blocking schemes, then it can be used as a dimension for scheme skylines in a scheme space and our scheme extension strategy can be applied. For example, reduction ratio (RR) has the following monotonic property, which is opposite to the monotomic property of PC, i.e. $RR(s_i) \geq RR(s_i \vee s_j)$ and $RR(s_i) \leq RR(s_i \wedge s_j)$. Consequently, an extension of conjunction helps increase RR values, but an extension of a disjunction can lead to lower RR values than before. Similarly, run time (RT) also has a monotonic property because it is monotonously increasing in terms of extensions, regardless whether an extension is a conjunction or a distinction, i.e. $RT(s_i) \leq RT(s_i \vee s_j)$ and $RT(s_i) \leq RT(s_i \wedge s_j)$.

4.2. Naive Skyline Learning

A naive way to learn skyline blocking schemes is to learn optimal blocking schemes under different thresholds in all dimensions. Then, based on these optimal blocking schemes, a scheme skyline can be obtained. In the following, we formulate the notion of optimal blocking scheme.

Definition 6. *(*Optimal blocking scheme*) Given a human oracle ζ, and a PC threshold $\epsilon \in [0, 1]$, an* optimal blocking scheme *is a blocking scheme $s_{\epsilon,\zeta}$ w.r.t. the following objective function, through selecting a training set T:*

$$\begin{aligned} &\textbf{\textit{maximize}} \quad PQ \\ &\textbf{\textit{subject to}} \quad PC \geq \epsilon, \text{ and } |T| \leq budget(\zeta) \end{aligned} \tag{7}$$

Let S be a subset of all possible blocking schemes. Then a blocking scheme s is called a *locally optimal blocking scheme* from S if $s \in S$ and it satisfies Eq. 7. Algorithm 1 describes a procedure of learning locally optimal blocking schemes using the active sampling strategy discussed in Section 3 and the scheme extension strategy discussed in Section 4.1. For clarity, we call this procedure the *Active Scheme Learning* (ASL) procedure.

In the following, we explain the key ideas of Algorithm 1. Let S be a set of blocking schemes, where each blocking scheme

Algorithm 1: Active Scheme Learning (ASL) Procedure

Input: Dataset: D
PC threshold $\epsilon \in (0, 1]$
Human oracle ζ with $budget(\zeta)$
Set of blocking predicates P
Sample size k
Output: A blocking scheme s

1 $S = S_{prev} = P, n = 0, T = \emptyset, X = \emptyset$
2 $X = X \cup$ RANDOM_SAMPLE(D)
3 **while** $n < budget(\zeta)$ **do**
4 **for** *each* $s_i \in S$ **do**
5 **if** $\gamma(s_i, X) \leq 0$ **then**
6 $X = X \cup$ SIMILAR_SAMPLE(D, s_i, k)
7 **else**
8 $X = X \cup$ DISSIMILAR_SAMPLE(D, s_i, k)
9 $n = |X|$
10 $T = T \cup \{(x_i, \zeta(x_i))|x_i \in X\}$
11 $s =$ FIND_OPTIMAL_SCHEME(S, T, ϵ); $S_{prev} = S$
12 **if** FOUND(s) **then**
13 $S = \{s \wedge s_i | s_i \in S_{prev}\}$
14 **else**
15 $s =$ FIND_APPROXIMATE_SCHEME(S, T, ϵ)
16 $S = \{s \vee s_i | s_i \in S_{prev}\}$
17 **Return** s

$s_i \in S$ is a blocking predicate at the beginning. The budget is initially set to zero, i.e. $n = 0$. A set of feature vectors is selected from the dataset D as seed samples (lines 1 and 2). Then, the procedure iterates until the number of samples in the training set reaches the budget limit (line 3). At the beginning of each iteration, the active sampling strategy is applied to generate a training set (lines 4 to 10). For each blocking scheme $s_i \in S$, the samples are selected in two steps: (1) firstly, the balance rate of this blocking scheme s_i is calculated (lines 5 and 7), (2) secondly, a feature vector to reduce this balance rate is selected from the dataset (lines 6 and 8). Then the samples are labeled by the human oracle and stored in the training set T. The label usage increases, accordingly (lines 9 and 10).

A locally optimal blocking scheme s is learned from S over the training set, according to a specified PC threshold ϵ (line 11). Then the scheme extension strategy is applied to determine whether a conjunction or disjunction form of this optimal blocking scheme and other blocking schemes will be used based on Lemma 1 (lines 12-16). If a locally optimal blocking scheme is found, new blocking schemes are generated by extending s to a conjunction with each of the blocking schemes in S_{prev}, so that the PQ values of new blocking schemes may be further increased (lines 12 and 13). Otherwise a blocking scheme with the maximum PC value is selected and new schemes are generated using disjunctions, so that the PC values of new schemes can be further increased (lines 14 to 16).

The *Naive Skyline Learning (Naive-Sky)* algorithm uses the ASL procedure to learn optimal blocking schemes under different PC thresholds. A high-level description for this algorithm is described in Algorithm 2. The input a user needs to specify is the label budget, the maximal ary of schemes in scheme skylines, and the size Δ of threshold interval, i.e. the difference of two consecutive thresholds. For example, if Δ is set to be 0.1, there will be in total 10 thresholds used, i.e. 0.1, 0.2, ..., 1.0, to learn at most 10 optimal blocking schemes. It is possible that the same blocking scheme is learned under two different thresholds. As shown in Algorithm 2, after initialization (line 1), this approach consists of two steps:

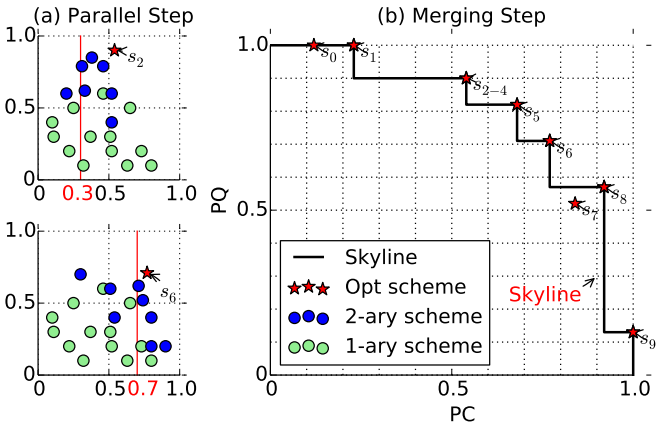

Figure 3: An illustration of the naive skyline learning algorithm, where optimal blocking schemes are learned in parallel as shown in (a), and the scheme skyline is depicted in (b), where s_7 is dominated by the skyline, and $s_2 - s_4$ are the same scheme shown as s_{2-4}.

- *Parallel step*: Optimal blocking schemes are learned in parallel under different thresholds using the ASL procedure (lines 2-5).
- *Merging step*: All optimal blocking schemes are merged and dominance between them is checked, leading to generating a scheme skyline (line 6).

Note that, in our Naive-Sky algorithm, the sample size k is uniformly decided by the total label budget $budget(\zeta)$, the threshold interval Δ, and the number of predicates $|P|$. Fig. 3(a) illustrates how two optimal blocking schemes are learned in parallel, where their thresholds are set to 0.3 and 0.7, respectively. In Fig. 3(b), optimal schemes that are learned under the thresholds 0.1, 0.2, ..., 1.0 are merged, and a scheme skyline is generated.

4.3. Adaptive Skyline Learning

Although the Naive-Sky algorithm offers the advantage of learning blocking schemes in parallel, it still has some limitations. For example, how to choose an appropriate threshold interval? If Δ is set too high, blocking schemes in a scheme skyline will be sparse and thus lack of accuracy. On the contrary, if Δ is set too low, users may have a large number of iterations being involved in learning blocking schemes under different thresholds, and some of them are redundant. This leads to unnecessary computational costs. A question arising naturally is whether the threshold interval Δ can be adjusted adaptively

Algorithm 2: Naive Skyline Learning (Naive-Sky)

Input: Dataset: D
Human oracle ζ with *budget*(ζ)
Set of blocking predicates P
Size of threshold interval Δ
Sample size k
Output: Scheme skyline S^*

1 $\epsilon = \Delta, S^* = \emptyset, S = \emptyset$
2 **while** $\epsilon \leq 1$ **do**
// Parallel step
3 $s = \text{ASL}(D, \epsilon, \zeta, P, k)$
4 $S = S \cup \{s\}$
5 $\epsilon += \Delta$
6 $S^* = \text{SKYLINE\_SCHEMES}(S)$ // Merging step
7 **Return** S^*

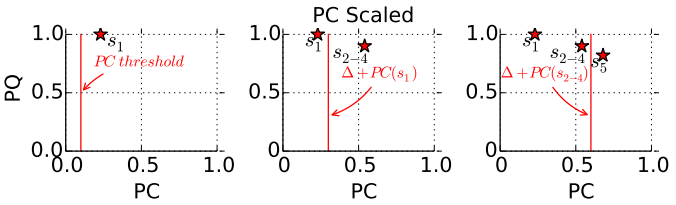

Figure 4: An illustration of adaptive skyline learning algorithm which can choose the PC threshold adaptively, based on the same example as in Fig. 3.

to improve the efficiency of algorithms but without sacrificing the accuracy of learned scheme skylines.

In our experiments with the Naive-Sky algorithm, we have noticed that the PC threshold specified by a user may not be (or even far from) the actual PC value of an optimal blocking scheme being learned.

Example 4. *Let us consider Fig. 3 again. In Fig. 3(a), when the threshold is set to 0.3, the actual PC value of the learned optimal blocking scheme is 0.53. Moreover, when the threshold is set to 0.4 and 0.5, e.g.* $\Delta = 0.1$*, the learned optimal blocking schemes remain unchanged, i.e., corresponding to the same point* s_{2-4} *shown in Fig. 3(b).*

Based on this observation, we propose another algorithm for learning scheme skylines, called *Adaptive Skyline Learning (Adap-Sky)*, which can adaptively learn a scheme skyline even for a small Δ. The following lemma formulates the property behind this observation.

Lemma 2. *Let* s_ϵ *and* $s_{\epsilon'}$ *be two optimal schemes learned using ASL under the PC thresholds* ϵ *and* ϵ'*, respectively, and under the same label budget. The following property holds:*

$$s_{\epsilon'} = s_\epsilon, \quad \forall \epsilon' \in [\epsilon, PC(s_\epsilon)] \tag{8}$$

where ϵ' *is a threshold from the range* $[\epsilon, PC(s_\epsilon)]$.

Proof 2. *Given a threshold* ϵ *and a set of blocking schemes* S *(*$\forall s \in S$*,* $PC(s) > \epsilon$*), we have* $s_\epsilon \in S$*. Given any threshold* ϵ'*, s.t.* $\epsilon < \epsilon' \leq PC(s_\epsilon)$*, and a set of blocking schemes* S' *(*$\forall s' \in S'$*,* $PC(s') > \epsilon'$ *), we thus have* $s_\epsilon \in S'$ *and* $s_{\epsilon'} \in S'$*. Based on Definition 6, the optimal blocking scheme is unique. This means* $s_\epsilon = s_{\epsilon'}$*.*

A high-level description of the Adap-Sky algorithm is presented in Algorithm 3. By Lemma 2, the Adap-Sky algorithm can adaptively select the next PC threshold based on the PC value of the current optimal blocking scheme (line 5).

Example 5. *Consider an illustration of the Adap-Sky algorithm shown in Fig. 4. In the leftmost plot, if* $\Delta = 0.1$*, the threshold* $\epsilon = 0.1$ *at the beginning, and the optimal blocking scheme is marked as* s_1*. In the middle plot, the new threshold is set to be* $PC(s_1) + 0.1$*, and the optimal blocking scheme is marked as* s_{2-4}*. This is different from Algorithm 2 which would take* $\epsilon = 0.1 + 0.1$ *as the new threshold. In the rightmost plot, which indicates the third iteration of the algorithm, the new threshold is* $PC(s_{2-4}) + 0.1$

Algorithm 3: Adaptive Skyline Learning (Adap-Sky)

Input: Dataset: D
Human oracle ζ with *budget*(ζ)
Set of blocking predicates P
Size of threshold interval Δ
Sample size k
Output: Scheme skyline S^*

1 $\epsilon = \Delta, S^* = \emptyset, S = \emptyset$
2 **while** $\epsilon \leq 1$ **do**
3 $s = \text{ASL}(D, \epsilon, \zeta, P, k)$
4 $S = S \cup \{s\}$
5 $\epsilon = \text{PC}(s) + \Delta$
6 $S^* = \text{SKYLINE\_SCHEMES}(S)$
7 **Return** S^*

4.4. Progressive Skyline Learning

The Adap-Sky algorithm has reduced the number of iterations needed in Naive-Sky. However, it still requires to construct a training set in each iteration, and these training sets are constructed independently. We notice that some samples used in one ASL procedure may also be used in other ASL procedures in different iterations. For example, if a sample is selected twice when PC is set to 0.3 and 0.7, its label is counted twice. This gives rise to a new question: can we further eliminate duplicate sampling occurring in the iterative process of constructing a training set so as to reduce the label cost?

To address the above question, we propose a *Progressive Skyline Learning* (Pro-Sky) algorithm. This algorithm can learn a scheme skyline without iteratively applying the ASL procedure. Instead, it learns a n-ary scheme skyline progressively, i.e., it starts by learning a 1-ary scheme skyline, then by learning a 2-ary scheme skyline, and finally by learning a n-ary scheme skyline, as illustrated in Fig 5. A user does not need to pre-define a threshold interval Δ.

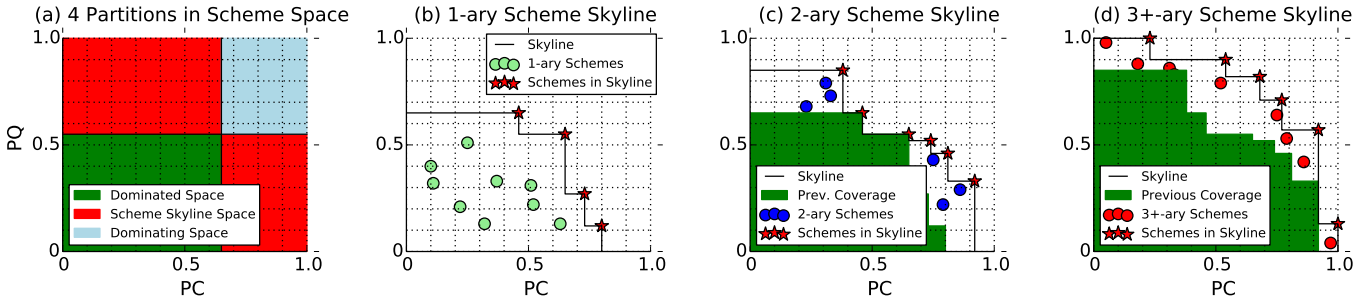

Figure 5: An illustration of progressive skyline learning algorithm: (a) shows three different spaces w.r.t. a blocking scheme in the crossing; (b) - (d) illustrate how our Pro-Sky algorithm may learn a scheme skyline progressively.

As shown in Fig. 5(a), given a blocking scheme in the crossing, we can partition a scheme space into four parts. Any blocking schemes mapped in the green area, called *Dominated Space*, are dominated by the given blocking scheme. The blocking schemes mapped in the red area, called *Scheme Skyline Space*, may possibly appear in a skyline. If a blocking scheme is mapped in the blue area, called *Dominating Space*, it will replace the given blocking scheme in the skyline. Therefore, our objective for progressive skyline learning is: given a scheme skyline S^*, we try to extend each $s \in S^*$ by discarding schemes in its dominated space, verifying schemes in its scheme skyline space, and finding schemes in its dominating space.

A high-level description for the Pro-Sky algorithm is described in Algorithm 4. The sampling steps (lines 1-11) remain the same as in the ASL procedure. However, when extending a scheme, this algorithm takes the property illustrated in Fig. 5(a) into consideration. Each scheme in a skyline is extended with all the predicates that are not contained by the scheme (lines 12-13). By conducting both conjunctions and disjunctions, we obtain both PC and PQ values of the new schemes, and select ones with incremental PC or PQ values (line 14). That is to say, if a new scheme fails into the red area as shown in Fig. 5(a), we would add this scheme into the candidate set. If it appears in the blue area, we would replace the previous one by this new scheme. If it appears in the green area, we would discard it.

Algorithm 4: Progressive Skyline Learning (Pro-Sky)

Input: Dataset: D
Human oracle ζ with $budget(\zeta)$
Set of blocking predicates P
Ary of schemes l
Output: Scheme skyline S^*

1 $S = P, n = 0, T = \emptyset, X = \emptyset, k = \frac{budget(\zeta)}{2l \times |P|}$
2 $X = X \cup \text{RANDOM\_SAMPLE}(D)$
3 **while** $n < budget(\zeta)$ **do**
4 **for** *each* $s_i \in S$ **do** `// Begin sampling`
5 **if** $\gamma(s_i, X) \leq 0$ **then**
6 $X = X \cup \text{SIMILAR\_SAMPLE}(D, s_i, k)$
7 **else**
8 $X = X \cup \text{DISSIMILAR\_SAMPLE}(D, s_i, k)$
9 $n = |X|$ `// End sampling`
10 $T = T \cup \{(x_i, \zeta(x_i)) | x_i \in X\}$ `// Add samples`
11 $S^* = \text{SCHEME\_SKYLINE}(S); S = \emptyset$
12 **for** $s_j \in S^*$ **do**
13 **for** $p_i \in P$ **do**
14 $\text{SELECT\_SCHEMES}(p_i, s_j, S)$
15 **Return** S^*

4.5. Complexity Analysis

In this section, we discuss the search complexity and the time complexity for learning the scheme skyline.

4.5.1. Search complexity

Given n blocking predicates, we have 2^n blocking schemes composed of n blocking predicates in conjunctions. Furthermore, if a blocking scheme is composed of at most n different blocking predicates (i.e. n-ary blocking scheme) in disjunction of conjunctions, we can regard them as the *monotonic boolean functions*, which are defined to be expressions combining the inputs (which may appear more than once) using only the operators conjunction and disjunction (in particular "not" is forbidden) [27]. Hence, the searching complexity for all possible blocking schemes can be $O(2^{\binom{n}{\lfloor n/2 \rfloor}})$ asymptotically, which is also known as the *Dedekind Number*. Learning a scheme skyline in this way is in high accuracy, because all blocking schemes will be considered. However, the scheme skyline learning would be space-consuming and label-inefficient, especially when the number of blocking predicates is large but the number of blocking schemes in a scheme skyline is small.

To analyze the search complexity of the algorithms Naive-Sky and Adap-Sky, we first analyze the complexity of ASL. Given n blocking predicates with a sufficient label budget, in the worst case, we can learn a n-ary blocking scheme as output. During the learning process, we first need to search for all n blocking schemes. Then based on the locally optimal one, we need to search for $n-1$ blocking schemes of 2-ary, and then, $n-2$ blocking schemes of 3-ary. Accordingly, the searching complexity of this ASL procedure is $O(n^2)$. Hence, for a given

Table 2: Characteristics of datasets

Datasets	# Attributes	# Records	# True Matches	Class Imbalance Ratio	# Blocking Predicates
Cora	4	1,295	17,184	1 : 49	16
DBLP-Scholar	4/4	2,616 / 64,263	2360	1 : 71,233	16/16
DBLP-ACM	4/4	2,616 / 2,294	2,224	1 : 2,698	16/16
NCVoter	18/18	6,233,785 / 6,981,877	6,122,579	$1 : 6.6 \times 10^6$	72/72
Amazon-GoogleProducts	4/4	1,363 / 3,226	1,291	1 : 3,405	20/20

Δ, the search complexity of Naive-Sky is $O(\frac{n^2}{\Delta})$, and in the worst case, the search complexity of Adap-Sky is the same as $O(\frac{n^2}{\Delta})$. Nonetheless, the search complexity of the algorithm Pro-Sky is different. If we use $|Opt_i|$ to describe the number of i-ary schemes being selected, then the search complexity of Pro-Sky will be $\sum_{i=1}^{n} |Opt_i| \times 2|P|$, where 2 indicates both conjunction and disjunction of schemes. In the worst case that all schemes are in the skyline, this is the same as *Dedekind Number*. However, if we further introduce a Δ in this algorithm, i.e., there should be at least a distance of Δ between two schemes in a skyline, the search complexity of Pro-Sky will be reduced to $\frac{n^2}{\Delta}$ ($\frac{1}{\Delta} \leq n$) in the worst case.

4.5.2. Time complexity

We further discuss the time complexity of sampling. To tackle the class imbalance problem, we select both similar and dissimilar samples w.r.t. a given blocking scheme s. However, as explained in Section 3, there may exist a high imbalance ratio between similar and dissimilar samples. In the worst case, we have to traverse the whole dataset to obtain one sample. Hence the time complexity to generate k samples is $O(|D| \times k)$ for one blocking predicate, where D is a dataset and k is the sample size for the blocking predicate.

To make the algorithms efficient under a large sample size k, we have implemented index tables for traditional blocking predicates as defined in the previous works [5, 24]. This implementation enables us to choose a similar or dissimilar sample in linear time. Therefore, the time complexity is reduced to $O(|D| + k)$ for a traditional blocking predicate.

Example 6. *Let us consider ⟨Authors, Same-soundex⟩ which is a blocking predicate previously discussed in Example 1. We can place all records into at most* 26×7 *buckets, where each bucket represents a soundex value such as G4. Hence, a match can be easily selected from the same bucket or a non-match can be selected from two different buckets. In this case, the time complexity is* $O(|D| + k)$.

In our work, block predicates are chosen by users. If a user chooses similarity-based blocking predicates, then the time complexity of selecting k samples will remain $|D| \times k$ for such a blocking predicate.

Remark 1. *In our experiments (will be further discussed in Section 5), we notice that traditional blocking predicates are sufficient to conduct blocking efficiently and effectively for most datasets. Only for Amazon-GoogleProducts, similarity-based blocking predicates may bring some benefits by trading off efficiency to certain degree. This also suggests that, by relaxing the definition of blocking predicate, our framework can be generalized to learn scheme skylines not only for blocking, but also for entity resolution itself, over various datasets.*

5. Experiments

We have evaluated our skyline learning algorithms to experimentally verify their performance. All our experiments have been run on a server with 12-core 64-bit Intel Xeon 2.4 GHz CPUs, 128GBytes of memory and Linux operating system.

5.1. Experimental Setup

In the following, we present the datasets, blocking predicates, baseline approaches, and measures used in our experiments.

5.1.1. Datasets

We have used five datasets in our experiments: (1) *Cora*[1] dataset contains bibliographic records of machine learning publications. (2) *DBLP-Scholar* [29] dataset contains bibliographic records from the DBLP and Google Scholar websites. (3) *DBLP-ACM* [29] dataset contains bibliographic records from the DBLP and ACM websites. (4) *NCVoter*[2] dataset contains real-world voter registration information of people from North Carolina in the USA. Two sets of records collected in October 2011 and December 2011 respectively are used in our experiments. (5) *Amazon-GoogleProducts* [29] dataset contains product entities from the online retailers Amazon.com and the product search service of Google accessible through the Google Base Data API. We summarize the characteristics of these data sets in Table 2. A complete list of attributes in these data sets are presented in Table 3.

5.1.2. Blocking predicates

We have used the following blocking functions in our experiments:

- *Same-value:* This blocking function takes two strings as input and compares their strings. The function returns 1 if the strings are exactly the same.

[1] Available from: *http://secondstring.sourceforge.net*
[2] Available from: *http://alt.ncsbe.gov/data/*

Table 3: Attributes of datasets

Datasets	Attributes
Cora	authors, pub_details, title, affiliation, conf_journal, location, publisher, year, pages, editors, appear, month
DBLP-Scholar	title, authors, venue and year
DBLP-ACM	
NCVoter	county_id, county_desc, voter_reg_num, voter_status_desc, voter_status_reason_desc, absent_ind, last_name, first_name, midl_name, full_name_rep, full_name_mail, reason_cd, status_cd, house_num, street_name, street_type_cd, res_city_desc and state_cd
Amazon-GoogleProducts	title/name, description, manufacturer, price

- *Same-soundex:* This blocking function takes two strings as input and transfers them into soundex codes based on a reference table [22]. It returns 1 if two codes are the same.
- *Same-doublemetaphone:* This blocking function also takes two strings as input and transfers each string into two codes from two reference tables [41]. It returns 1 if either code is the same. Compared with *Same-soundex*, it performs better when being applied to Asian names.
- *Get-substring:* This blocking function takes only a segment (e.g. first four letters) of the whole string for comparison, and returns 1 if the segments from two strings are the same.
- *Top-similarity:* This blocking function takes two strings as input and returns 1 if one string is most similar to the other; otherwise, return 0.

The first four blocking functions have been applied to all attributes in the datasets depicted in Table 3, which accordingly leads to 16 or 72 different blocking predicates for datasets as shown in Table 2. The last blocking function has only been used in the dataset Amazon-GoogleProducts due to two reasons: (1) it has a higher time complexity, and (2) it has little effects on the other datasets.

5.1.3. Baseline approaches

We have used the following approaches as the baselines: (1) *Fisher* [24], which is the state-of-the-art unsupervised scheme learning approach proposed by Kejriwal and Miranker. Details of this approach will be outlined in Section 6. (2) *TBlo* [19], which is a traditional blocking approach based on expert-selected attributes. In the survey [14], this approach has a better performance than the other approaches in terms of the F-measure results. (3) *RSL (Random Scheme Learning)*, which is an algorithm that has the same structure of the ASL procedure except that random sampling is used to build a training set, instead of active sampling. In each experiment, we have run the RSL ten times. We present the average results of the blocking schemes it has learned. As reported in [24], Fisher has shown a better performance compared with the supervised blocking scheme learning approach [5]. We thus do not consider it as a baseline.

5.1.4. Measures

We use the following measures [14] to evaluate the blocking quality. *Reduction Ratio (RR)* is one minus the total number of record pairs in blocks divided by the total number of record pairs without blocks, i.e., RR measures the reduction of the number of compared pairs with and without blocks. *Pairs Completeness (PC)* and *Pairs Quality (PQ)* have been defined in Section 2. *F-measure* (FM) = $\frac{2*PC*PQ}{PC+PQ}$ is the harmonic mean of PC and PQ.

The size of a training set directly affects the results of scheme skyline learning. If a training set is small, an algorithm may learn different skylines in different runs. This is because the undersampling can lead to biased samples which do not represent the characteristics of the whole sampling space. Hence, we define the notion of *constraint satisfaction* as $CS = \frac{N_s}{N}$ to describe the learning stability of an algorithm, where N_s denotes the number of times that an algorithm can learn a blocking scheme, and N is the total number of times the algorithm runs. For example, if an algorithm runs ten times and learns three different scheme skylines with 2, 3, and 5 times, respectively. Then the CS values for these three blocking schemes are $\frac{2}{10} = 0.2$, $\frac{3}{10} = 0.3$ and $\frac{5}{10} = 0.5$, respectively, in this case.

In the rest of this section, we will discuss the experimental results of our skyline algorithms, and compare the performance of our algorithms with the baseline approaches.

5.2. Label Efficiency

We first have conducted experiments to evaluate the label efficiency of our algorithms.

5.2.1. Label costs

The label costs of our proposed algorithms, namely *Naive-Sky* for algorithm 2, *Adap-Sky* for algorithm 3 and *Pro-Sky* for algorithm 4, are shown in Table 4, where the label costs of these algorithms for learning scheme skylines of five datasets with $CS = 90\%$ are recorded. We consider two variants of *Naive-Sky*: sequential *Naive-Sky* and parallel *Naive-Sky*. Sequential *Naive-Sky* iteratively computes optimal blocking schemes with the PC threshold being increased by Δ in each iteration. Parallel *Naive-Sky* computes a number of optimal blocking schemes in parallel, where a range of thresholds from 0 to 1 with interval Δ are assigned to these parallel computations (one threshold for each computation). We set $\Delta = 0.1$ and $\Delta - 0.05$ for all datasets. For example, if $\Delta = 0.1$, sequential *Naive-Sky* iterates 10 times with the PC threshold being increased by 0.1 at each time, and parallel *Naive-Sky* takes 10 parallel computations, each computation deals with the ASL procedure once with a PC threshold ranging from 0.1 to 1. Furthermore, for *Naive-Sky* and *Adap-Sky*, the label budget begins with 50, and increases by 50 for each run of ASL. The total label cost is accumulated when the PC threshold increases. For *Pro-Sky*, the

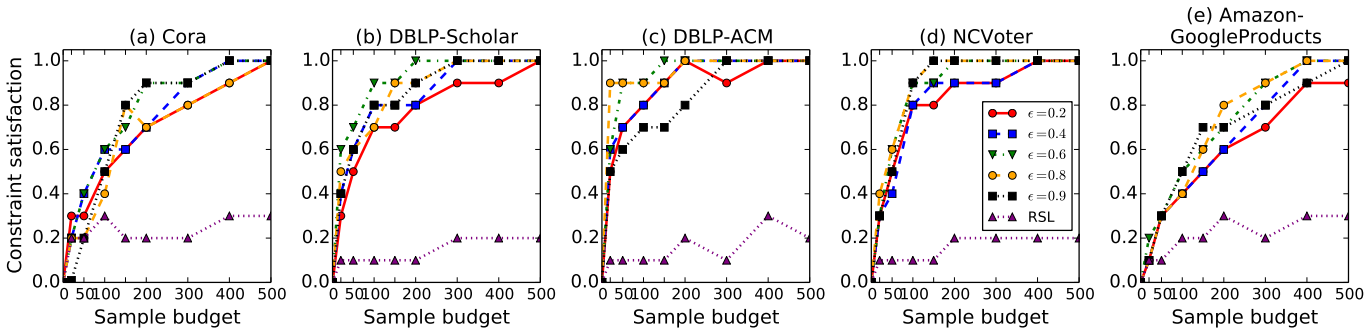

Figure 6: Comparison on constraint satisfaction by ASL and RSL under different label budgets over five datasets

label budget begins with 500, and increases by 500. This is to ensure a fair comparison with *Naive-Sky* and *Adap-Sky* because, when Δ is set to 0.1, *Naive-Sky* needs exactly ten iterations to reach 1 from 0.1, and *Adap-Sky* needs at most ten iterations to reach 1 from 0.1. Thus, the total label consumption of these two algorithms is $50 \times 10 = 500$ in the worst case. Since *Pro-Sky* does not involve iterations, we set its sample size to 500. The label cost is recorded when its CS value reaches 90% in ten consecutive runs. Table 4 shows that *Adap-Sky* saves more labels than *Naive-Sky*; nonetheless, *Pro-Sky* can reduce the label cost from one third to a half of the label cost of *Adap-Sky*.

Both *Naive-Sky* and *Adap-Sky* use ASL to learn scheme skylines. We have thus evaluated the label efficiency of the ASL procedure. In Table 5, we present the label costs for learning a stable blocking scheme with CS = 90% under different PC thresholds, and compare them with the number of labels required by RSL (i.e. the algorithm based on random sampling). The minimum label cost for each dataset is highlighted. In our experiments, the label budget for both ASL and RSL under a given PC threshold begins with 50, and is then increased by 50 each time. The experiments for both ASL and RSL terminate when the CS values of the learned blocking schemes reach 90% in ten consecutive runs.

5.2.2. *Constraint satisfaction evaluation*

To further analyze the label efficiency of our algorithms, especially *Naive-Sky* and *Adap-Sky*, we have monitored CS values under different label budgets (ranging from 20 to 500) in terms of various PC thresholds, i.e. $\epsilon \in \{0.1, 0.2, 0.4, 0.6, 0.8\}$, over five datasets. The results are presented in Figure 6.

To make a fair comparison on active sampling and random sampling, the number of samples for training RSL is equal to the label budget used in the ASL procedure. Our experimental results show that random sampling with a limited number of labels fails to identify an optimal blocking scheme. Additionally, both PC threshold and label budget can affect CS values. In general, when the label budget increases or the PC threshold ϵ decreases, the CS value grows. It is worthy to note that an extremely high PC threshold is usually harder to achieve, and thus sometimes no blocking scheme can be learned due to the limitation of samples (e.g. the black line with square marks under 100 sample budget). On the other hand, when the label budget is extremely low and the PC threshold is also low (e.g. the red line with $\epsilon = 0.2$), the algorithms may generate a large number of different blocking schemes satisfying the PC threshold, leading to a low CS value.

5.2.3. *Label efficiency analysis*

We have also analyzed the factors that may affect the label costs. First, as explained before, both extremely high and low PC thresholds may require more labels than in other cases. Second, datasets of a smaller size (i.e., containing a smaller number of record pairs) often need less labels. Furthermore, datasets with more attributes have a larger number of blocking predicates and thus need a higher label budget in order to be able to consider all blocking predicates in sampling. The quality of a dataset may also affect the label costs. Generally, the cleaner a dataset is, the less labels it requires. For example, although Cora has the smallest number of attributes and records, it still needs the largest number of labels because it contains many fuzzy values (e.g. mis-spelling names, first name and last name being swapped and etc.). On the contrary, NCVoter needs a lower label budget although it has more attributes and records than several other datasets. The cleanness of a dataset also affects the distribution of label costs under different thresholds. For example, with a PC threshold $\epsilon = 0.2$, we need only 200 labels for NCVoter but 550 labels for Cora to generate blocking schemes with CS = 90%.

5.3. *Time Efficiency*

Table 4 shows Run Time (RT) of our three algorithms over five datasets, in relation to different label budgets. Two threshold intervals are used, i.e. $\Delta = 0.05$ and $\Delta = 0.1$. Generally speaking, the RT values depend on two factors: (1) the size of a dataset and (2) the number of samples. With the increase of the label budget and the number of records in a dataset, the RT value increases. We can also see that, for these three algorithms, the threshold interval Δ does affect the run time, but not significantly.

We also present the RT values of *Naive-Sky* in two different settings: running sequentially and running in parallel. The

[3] A pre-calculated similarity matrix is used for searching most similar records; hence, RT does not consider the time consumption of similarity computation.

Table 4: Comparison on the label costs and run times (in seconds) of three skyline algorithms over five datasets

	Cora			DBLP-Scholar			DBLP-ACM			NCVoter			Amazon-GoogleProducts		
	Budget	Δ	RT	Budget	Δ	RT	Budget	Δ	RT	Budget	Δ	RT	Budget	Δ	RT[3]
Naive-Sky (Sequential)	6000	0.1	18.74	5000	0.1	96.71	3000	0.1	76.95	3500	0.1	225.08	5000	0.1	98.3
		0.05	21.55		0.05	133.25		0.05	90.8		0.05	267.85		0.05	120.47
Naive-Sky (Parallel)	6000	0.1	12.19	5000	0.1	35.51	3000	0.1	50.6	3500	0.1	83.92	5000	0.1	72.17
		0.05	18.33		0.05	40.5		0.05	73.51		0.05	110.81		0.05	93.68
Adap-Sky	4200	0.1	12.48	3500	0.1	56.28	1250	0.1	33.23	1400	0.1	118.4	1600	0.1	48.93
		0.05	14.68		0.05	60.73		0.05	34.55		0.05	122.16		0.05	60.36
Pro-Sky	2500	0.1	5.6	2000	0.1	32.56	1000	0.1	11.38	1000	0.1	63.09	1000	0.1	17.33
		0.05	6.27		0.05	38.09		0.05	13.28		0.05	70.56		0.05	20.1

Table 5: Comparison on the label costs of ASL and RSL with CS = 90%

ϵ	Cora	DBLP-Scholar	DBLP-ACM	NCVoter	Amazon-GoogleProducts
0.2	600	500	300	300	400
0.4	**400**	350	200	350	400
0.6	450	**250**	**150**	250	250
0.8	550	300	200	**200**	**200**
0.9	500	**250**	300	250	300
RSL	7,900	10,000+	2,200	10,000+	5,000

parallel method can reduce the run time considerably, varying from 30% to 70%. Especially, for the DBLP-Scholar and NCVoter datasets, *Naive-Sky* running in parallel can even outperform *Adap-Sky*.

5.4. Blocking Quality

Now we discuss how the blocking quality of our skyline algorithms may be affected by the choice of n-ary blocking schemes, as well as PC thresholds. We also compare the quality of blocks generated by our skyline algorithms with the quality of blocks generated by the baseline approaches.

5.4.1. Under different number of aries

The experimental results of the *Pro-Sky* algorithm under different number of aries are presented in Figure 7. We have also tested on *Naive-Sky* and *Adap-Sky* with $\Delta = 0.05$ and $\Delta = 0.10$. However, because the blocking schemes in the scheme skylines learned by these algorithms are the subsets of the scheme skyline generated by *Pro-Sky* in which no Δ is defined, we thus omit the results for *Naive-Sky* and *Adap-Sky*. Figure 7 illustrates how *Pro-Sky* can progressively generate the scheme skylines using blocking schemes from 1-ary to 3+-ary over five datasets. We do not present the further process of 3+-ary, as the scheme skylines have already been generated within 3-ary and they would remain the same. In the plots (d2) and (d3) of Figure 7, we present the scheme skylines with $PC \in [0.995, 1]$ because most of the schemes on the skyline for NCVoter dataset are located in this range.

The *Pro-Sky* algorithm can detect a number of 1-ary schemes and generate the scheme skylines as shown in the plots (a1), (b1), (c1) and (d1) of Figure 7, although some of the learned blocking schemes are not on the skylines. When considering 2-ary blocking schemes in both conjunction and disjunction, we notice that disjunction schemes have higher PC values but lower PQ values compared with conjunction ones. Some 2-ary blocking schemes have better performance than 1-ary blocking schemes in the scheme skylines but some are not. Hence, a scheme skyline can be updated with some segments remaining the same, e.g. PC values from 0.4 to 0.63 in the plot (b2). The third row of the figure shows 3+-ary blocking schemes. The scheme skylines (red dashed lines) do not change much compared with the 2-ary scheme skylines (blue dashed lines). We also show the results of the baseline approaches, where Fisher normally generates the blocking schemes with high PC values, but TBlo schemes are not in a skyline in the most cases.

5.4.2. Under different PC thresholds

To make a detailed comparison with the baseline approaches, we have conducted experiments based on the ASL procedure under different PC thresholds. We have monitored the blocking schemes learned by the ASL procedure as well as their PC and PQ values under different PC thresholds $\epsilon \in \{0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$. The results are presented in Figure 8. We can see that the PC and PQ values may vary because the learned blocking schemes are different when the PC thresholds increase. The label budget is ensured to be sufficient in these cases.

With the increment of the PC thresholds, the blocking schemes we learn generate lower PC values and higher PQ values. However, there are still some overlapping points. This indicates that under certain thresholds, the *Pro-Sky* algorithm learns the same blocking scheme. On the other hand, this also provides further evidence showing the efficiency of *Adap-Sky* compared with *Naive-Sky*. For example, the performance of *Pro-Sky* for NCVoter dataset is consistent with both high PC and PQ values, whatever the PC threshold is. On the contrary, for other datasets such as Cora, a lower PC threshold allows the *Pro-Sky* algorithm to seek for blocking schemes that can generate higher PQ values, but the PC values decrease. We also notice that, for DBLP-ACM and NCVoter datasets, blocking schemes with both high PC and PQ values are learned with a low threshold (e.g. $\epsilon = 0.4$), but for DBLP-Scholar, no blocking schemes can be learned with high PQ values (i.e. higher than 0.6). In the figure, the blocking schemes with the PC threshold 1.0 are normally hard to learn, hence we present the maximum PC values of the blocking schemes learned by our algorithms.

5.4.3. Compared with baselines

Existing work largely focuses on learning a single blocking scheme, while our algorithms aim to learn a scheme skyline, which is a set of blocking schemes. Hence we first conduct experiments to show that, in a 2-dimension space of PC and

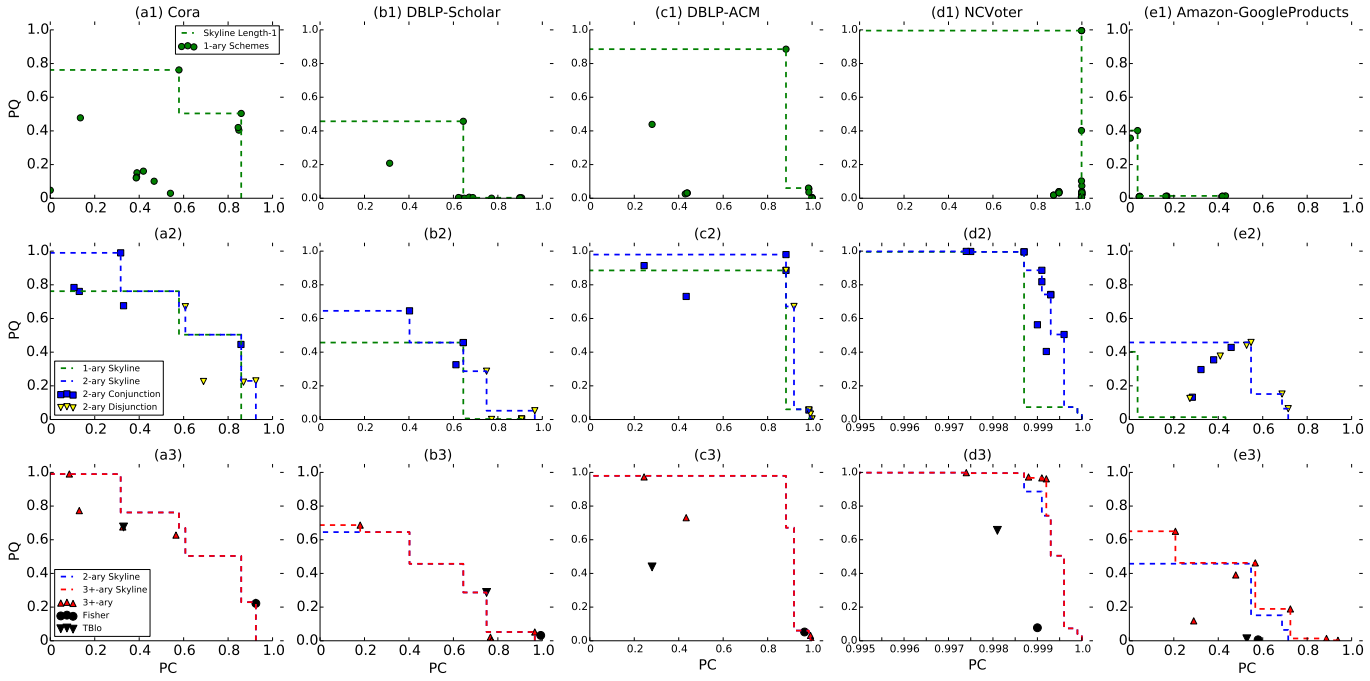

Figure 7: An illustration of the progressive process for learning scheme skylines by *Pro-Sky* over five datasets. The three rows from top to bottom show the results of 1-ary, 2-ary (in both conjunction and disjunction) and 3-ary blocking schemes, respectively.

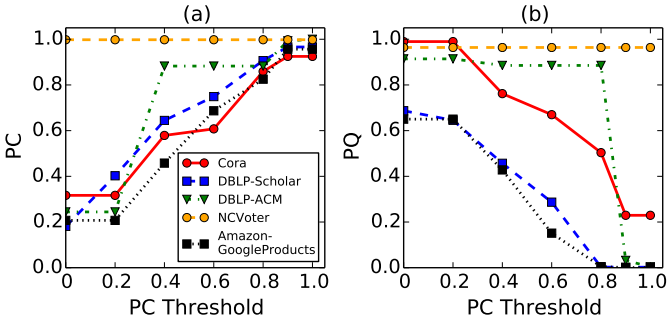

Figure 8: Quality of blocks generated by blocking schemes using *Pro-Sky* under different PC thresholds over five datasets: (a) PC and (b) PQ

PQ, the blocking schemes learned by the baselines *TBlo* and *Fisher* are dominated by or contained in our scheme skylines over five datasets. The experimental results are shown in the plots (a3), (b3), (c3), and (d3) of Figure 7.

To make a fair point-to-point comparison with the baseline approaches, we have further conducted experiments which consider the baseline PC values as the PC thresholds for the ASL procedure. Table 6 presents the PC and PQ values in the experimental results. Compared with *TBlo*, our *Pro-Sky* algorithm can generate blocks with much higher PQ values (i.e. from 50% to 101%) while remaining similar PC values, except in *DBLP-Scholar* where the results are the same. Compared with *Fisher*, in the most cases, the results are the same, except in *DBLP-Scholar*, the results generated by our *Pro-Sky* algorithm have a 12 times higher PQ value. In general, our *Pro-Sky* algorithm

Table 6: Comparison on blocking quality where the baseline PC values are selected as thresholds

	PC		PQ	
	TBlo	ASL	TBlo	ASL
Cora	0.3296	0.3167	0.6758	0.9898
DBLP-Scholar	0.7492	0.7492	0.2869	0.2869
DBLP-ACM	0.2801	0.8826	0.4387	0.8854
NCVoter	0.9981	0.9979	0.6558	0.9640
Amazon-GoogleProducts	0.5285	0.5665	0.0118	0.4627

	PC		PQ	
	Fisher	ASL	Fisher	ASL
Cora	0.9249	0.9249	0.2219	0.2219
DBLP-Scholar	0.9928	0.9928	0.0320	0.0320
DBLP-ACM	0.9661	0.9686	0.0522	0.6714
NCVoter	0.9990	0.9990	0.0774	0.0774
Amazon-GoogleProducts	0.5792	0.7244	0.0059	0.1892

can generate results as good as or better than the baseline approaches under the same thresholds and with a sufficient label budget.

We also present blocking schemes with the highest f-measure values learned by our algorithms and the baselines, and compare their FM, PC and PQ values. The FM results are shown in Figure 9(a) in which our *Pro-Sky* algorithm outperforms all the baseline approaches over all the datasets. Since all the ap-

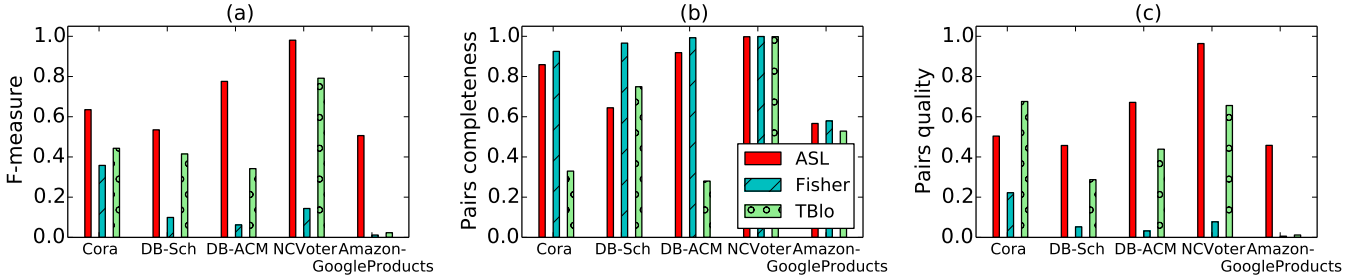

Figure 9: Comparison on blocking quality by using different blocking approaches over five datasets and using the measures: (a) FM, (b) PC, and (c) PQ

proaches yield high RR values over five datasets, the RR values are not presented in the figure. In Figure 9(b), the PC values of our *Pro-Sky* algorithm are not the highest ones over the five datasets, but they are not much lower than the highest ones (i.e. within 10% lower except in DBLP-Scholar). Moreover, our *Pro-Sky* algorithm can generate higher PQ values than all the other approaches, from 15 percents higher in NCVoter (0.9956 vs 0.8655) to 20 times higher in DBLP-ACM (0.6714 vs 0.0320), as shown in Figure 9(c).

6. Related Work

Scheme based blocking techniques for entity resolution were first mentioned by Fellegi and Sunter [19]. They used a pair *(attribute, matching-method)* to define blocks. For example, the soundex code of both names *Gail* and *Gayle* is "G4", thus records that contain either of the names are placed into the same block. At that time, a blocking scheme was chosen by domain experts without using any learning algorithm. Later on, a number of scheme learning approaches have been proposed, which generally fall into two categories: (1) supervised blocking scheme learning approaches [32, 7], and (2) unsupervised blocking scheme learning approaches [24, 25, 26].

Michelson and Knoblock [32] proposed a supervised blocking scheme learning algorithm, called *Blocking Scheme Learner*, which is the first supervised algorithm to learn blocking schemes. This approach adopts the Sequential Covering Algorithm(SCA) [33] to learn blocking schemes aiming to achieve both high PC values and high RR values. In the same year, Bilenko et al. [5] proposed two blocking scheme learning algorithms called *ApproxRBSetCover* and *ApproxDNF* to learn disjunctive blocking schemes and DNF (i.e. Disjunctive Normal Form) blocking schemes, respectively. Both algorithms are supervised and need training data. The former algorithm focused on using conjunction and disjunction to balance RR and PC values, while the later algorithm considered constructing blocking schemes from a set of predicates in a disjunctive form. Then, Cao et al. [7] noticed that obtaining labels for training data is very expensive, and thus used both labeled and unlabeled samples in their approach. Their algorithm can learn a blocking scheme using conjunctions of blocking predicates to satisfy both minimum true-match coverage and minimum precision criteria.

Later on, Kejriwal et al. [24] proposed an unsupervised algorithm for learning blocking schemes, where no label from a human oracle is needed. Instead, a weak training set was applied, where both positive and negative labels were generated by calculating the similarity of two records in terms of TF-IDF. All the candidate predicates are sorting by the fisher score. The predicate with the highest score is selected as a component of the final blocking scheme, and if lower ranking predicates can cover more positively labeled pairs in the training set, they will be selected in a disjunctive form. After traversing all the predicates, a blocking scheme is learned. Although this approach circumvents the need of human oracles, using unlabeled samples or labeled samples only based on string similarity is not reliable [47] and hence blocking quality can not be guaranteed [48].

There are also some schema-agnostic blocking approaches: (1) Token based blocking, which is also called Q-gram based blocking, aims to group all the records containing the same token (string of length q) into the same block [2, 36]. (2) Clustering based blocking [37] aims to group a number of records into the same block by pre-defined criterion using some computationally cheap clustering techniques. These criterion can be similarity thresholds [31], a number of k nearest neighbors [15] and so on. (3) One of the hot topics on blocking is meta-blocking, which aims to discard the redundant records which are high likely to be superfluous from all blocks, so that the comparison time will reduce. A number of strategies have been mentioned in recent work [38, 45].

Until now, all these related approaches for ER blocking focused on learning a single blocking scheme under given constraints. No work has been reported to provide an overview of all possible blocking schemes under different constraints for users so that they can choose their preferred blocking schemes under their own constraints. To fill in this gap, in this paper, we consider to use skyline techniques to present a set of optimal blocking schemes satisfying different requirements.

Skyline queries have been widely studied in the context of databases. Skyline queries can be regarded as queries of value-for-money [23], which provides a set of options representing the optimal combinations of the characteristics of the database, for example, the location and price of all the hotels in a city. A good number of approaches have been developed in recent years [10, 30, 46, 42], which primarily focused on learning representative skylines, such as top-k RSP (representative skyline points) [30], k-center (i.e. choose k centers and one skyline point for each center) [46] and threshold-based preference [42].

From an algorithmic perspective, a naive algorithm for skyline queries (e.g., nested-loop algorithm) has the time complexity $O(n^2d)$, where n is the number of records and d is the number of attributes in a given database. Later, several algorithms have been proposed to improve the efficiency of skyline queries based on different properties which have been previously ignored by the naive algorithm. In the early days, Borzsony et al. [6] proposed the BNL (block nested-loop) algorithm based on the transitivity of the dominance relation (e.g. if a dominates b and b dominates c, then a dominates c) . Then, Chomicki et al. [11, 12] proposed an SFS (sort-filter-skyline) algorithm with the improvements: progressive and optimal comparison times. Sheng and Tao proposed an EM (external memory) model based on the attribute order[6]. Morse et al. proposed the LS-B (lattice skyline) algorithm based on the low cardinality of some attributes (e.g. the rating of a movie is an integer within a small range of [1, 5]) [34]. Papadias et al. proposed the BBS (branch-and-bound skyline) algorithm based on the index of all input records by an R-tree [40].

Compared with our approach, existing works on skyline queries aimed to efficiently tease out a skyline of queries over a database in which records and attributes are known. In contrast, our study in this paper has shifted the focus to learning a skyline of blocking schemes in terms of a given number of selection criteria but the actual values of those selection criteria are not available in a database. Particularly, in many real-world applications, only a limited number of labels are allowed to be used for assessing blocking schemes. Thus, how to efficiently and effectively learn a skyline of blocking schemes is a difficult task. In this paper, we consider to leverage active learning techniques for finding informative samples and improving the performance of learning.

Active learning has been extensively studied in the past [43]. Ertekin et al. [18] showed that active learning may provide almost the same or even better results in solving the class imbalance problem, in comparison with the oversampling and/or undersampling approaches [8]. In recent years, a number of active learning approaches have been introduced for entity resolution [1, 3, 20]. For example, Arasu et al. [1] proposed an active learning algorithm based on the monotonicity assumption, i.e. the more textually similar a pair of records is, the more likely it is a matched pair. Their algorithm aimed to maximize recall under a manually specified precision constraint. To reduce the label and computational complexity, Bellare et al. [3] proposed an approach to solve the maximal recall with precision constraint problem by converting this problem into a classifier learning problem. In their approach, the main algorithm is called *ConvexHull* which aimed to find the best classifier with the maximal recall by finding the classifier with the minimal 01-Loss. Here the 01-Loss refers to the total number of false negatives and false positives. The authors then designed another algorithm called *RejectionSampling* which used a black-box to generate the 01-Loss of each classifier. The black-box invokes the *IWAL* algorithm in [4].

In this paper, for the first time, we study the problem of scheme skyline learning. Previous works aimed to learn a single blocking scheme that matches user requirements. Skyline techniques may help to search for skylines of queries, but traditionally, they regard all the attribute values as known. Different from previous works on blocking schemes and skyline queries, we propose novel algorithms to efficiently learn skylines of blocking schemes in this paper.

7. Conclusions and Future Work

In this paper, we have proposed a scheme skyline learning framework called skyblocking, which integrates skyline query techniques and active learning techniques into learning a set of optimal blocking schemes under different constraints and a limited label budget. We have tackled the class imbalance problem by solving the balanced sampling problem. We have also proposed the scheme extension strategy to reduce the searching space and label costs. We have further developed three algorithms for efficiently learning scheme skylines. Our algorithms overcome the weaknesses of existing blocking scheme learning approaches in two aspects: (1) Previous supervised blocking scheme learning approaches require a large number of labels for learning a blocking scheme, which is an expensive task for entity resolution; (2) Existing unsupervised blocking scheme learning approaches generate training sets based on the similarity of record pairs, instead of true labels, thus the training quality can not be guaranteed.

Skyline query is a useful technique for decision making. In the future, we may consider an extended active learning framework which will not be limited to the blocking scheme learning, but can also be applied for schema selection and so on. Furthermore, we may consider some other state-of-the-art techniques for sampling from imbalanced data to deal with the blocking scheme learning problem in entity resolution over imbalanced datasets.

Acknowledgment

This work was partially funded by the Australian Research Council (ARC) under Discovery Project DP160101934.

References

[1] A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *SIGMOD*, pages 783–794. ACM, 2010.

[2] L. R. Baxter, R. Baxter, P. Christen, et al. A comparison of fast blocking methods for record. 2003.

[3] K. Bellare, S. Iyengar, A. G. Parameswaran, and V. Rastogi. Active sampling for entity matching. In *SIGKDD*, pages 1131–1139. ACM, 2012.

[4] A. Beygelzimer, D. J. Hsu, J. Langford, and T. Zhang. Agnostic active learning without constraints. In *ANIPS*, pages 199–207, 2010.

[5] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, pages 87–96. IEEE, 2006.

[6] S. Borzsony, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430. IEEE, 2001.

[7] Y. Cao, Z. Chen, J. Zhu, P. Yue, C.-Y. Lin, and Y. Yu. Leveraging unlabeled data to scale blocking for record linkage. In *IJCAI*, volume 22, page 2211, 2011.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[9] S. Chester and I. Assent. Explanations for skyline query results. In *EDBT*, pages 349–360, 2015.

[10] J. Chomicki, P. Ciaccia, and N. Meneghetti. Skyline queries, front and back. *SIGMOD Record*, 42(3):6–18, 2013.

[11] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719. IEEE, 2003.

[12] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting: Theory and optimizations. In *IIPWM*, pages 595–604. Springer, 2005.

[13] P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

[14] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *TKDE*, 24(9):1537–1555, 2012.

[15] P. Christen et al. Towards parameter-free blocking for scalable record linkage. 2007.

[16] P. Christen, D. Vatsalan, and Q. Wang. Efficient entity resolution with adaptive and interactive training data selection. In *ICDM*, pages 727–732. IEEE, 2015.

[17] U. Draisbach and F. Naumann. A comparison and generalization of blocking and windowing algorithms for duplicate detection. In *International Workshop on QDB*, pages 51–56, 2009.

[18] S. Ertekin, J. Huang, L. Bottou, and L. Giles. Learning on the border: active learning in imbalanced data classification. In *CIKM*, pages 127–136. ACM, 2007.

[19] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[20] J. Fisher, P. Christen, and Q. Wang. Active learning based entity resolution using markov logic. In *PAKDD*, pages 338–349. Springer, 2016.

[21] J. Fisher, P. Christen, Q. Wang, and E. Rahm. A clustering-based framework to control block sizes for entity resolution. In *SIGKDD*, pages 279–288. ACM, 2015.

[22] D. Holmes and M. C. McCabe. Improving precision and recall for soundex retrieval. In *ITCC*, pages 22–26. IEEE, 2002.

[23] C. Kalyvas and T. Tzouramanis. A survey of skyline query processing. *arXiv preprint arXiv:1704.01788*, 2017.

[24] M. Kejriwal and D. P. Miranker. An unsupervised algorithm for learning blocking schemes. In *ICDM*, pages 340–349. IEEE, 2013.

[25] M. Kejriwal and D. P. Miranker. A two-step blocking scheme learner for scalable link discovery. In *OM*, pages 49–60, 2014.

[26] M. Kejriwal and D. P. Miranker. A dnf blocking scheme learner for heterogeneous datasets. *arXiv preprint arXiv:1501.01694*, 2015.

[27] A. Kisielewicz. A solution of dedekincts problem on the number of isotone boolean functions. *J. reine angew. math*, 386:139–144, 1988.

[28] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.

[29] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *VLDB Endowment*, 3(1-2):484–493, 2010.

[30] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDM*, pages 86–95. IEEE, 2007.

[31] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. Citeseer, 2000.

[32] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.

[33] T. M. Mitchell et al. Machine learning. wcb, 1997.

[34] M. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, pages 267–278. VLDB Endowment, 2007.

[35] K. OHare, A. Jurek, and C. de Campos. A new technique of selecting an optimal blocking method for better record linkage. *Information Systems*, 77:151–166, 2018.

[36] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 535–544. ACM, 2011.

[37] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederee, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2665–2682, 2012.

[38] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl. Meta-blocking: Taking entity resolutionto the next level. *TKDE*, 26(8):1946–1960, 2014.

[39] G. Papadakis, G. Papastefanatos, and G. Koutrika. Supervised meta-blocking. *VLDB Endowment*, 7(14):1929–1940, 2014.

[40] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478. ACM, 2003.

[41] L. Philips. The double metaphone search algorithm. *C/C++ users journal*, 18(6):38–43, 2000.

[42] A. D. Sarma, A. Lall, D. Nanongkai, R. J. Lipton, and J. Xu. Representative skylines using threshold-based preference distributions. In *ICDE*, pages 387–398. IEEE, 2011.

[43] B. Settles. Active learning. *Synthesis Lectures on AIML*, 6(1):1–114, 2012.

[44] J. Shao and W. Qing. Active blocking scheme learning for entity resolution. In *PAKDD*, 2018.

[45] G. Simonini, S. Bergamaschi, and H. Jagadish. Blast: a loosely schema-aware meta-blocking approach for entity resolution. *VLDB Endowment*, 9(12):1173–1184, 2016.

[46] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, pages 892–903. IEEE, 2009.

[47] Q. Wang, M. Cui, and H. Liang. Semantic-aware blocking for entity resolution. *TKDE*, 28(1):166–180, 2016.

[48] Q. Wang, J. Gao, and P. Christen. A clustering-based framework for incrementally repairing entity resolution. In *PAKDD*, pages 283–295. Springer, 2016.

[49] Q. Wang, D. Vatsalan, and P. Christen. Efficient interactive training selection for large-scale entity resolution. In *PAKDD*, pages 562–573. Springer, 2015.